Reproduced from an old troff file (dating 1987/88).

COMPUTATIONAL RANDOMNESS

(a survey)

Oded Goldreich

ABSTRACT

A recent behaviouristic approach to randomness is surveyed. In this approach a probability distribution is considered "pseudorandom" if no "efficient procedure" can distinguish it from the uniform probability distribution. Remarkably, pseudorandomness so defined is expandable in the sense that (assuming the existence of 1-1 one-way functions) short pseudorandom sequences can be deterministically and efficiently expanded into much longer pseudorandom sequences. The new approach to randomness is based on basic concepts and results from the theory of resource-bounded computation. In order to make the survey as accessible as possible, we have presented elements of the theory of resource bounded computation (but only to the extent required for the description of the new approaches). This survey is not intended to provide an account of the more traditional approaches to randomness (e.g. Kolmogorov Complexity) and this approach is described only in order to confront it with the new approach.

Note added in Dec. 1997: Section 4 of this survey may be read only for historical purposes. Otherwise, it is far inferior to results known nowadays, and consequently to existing presentations. For further details see Appendix.

1. INTRODUCTION

Randomness is playing an increasingly important role in computation. It is frequently used in the design of sequential, parallel and distributed algorithms, and is of course central to cryptography. Gaining a better understanding of randomness thus yields a deeper understanding of computation and stronger computational tools. Indeed, a lot of attention has been devoted to formalize the notion of randomness and for building good pseudo-random number generators. Kolmogorov suggested to measure the randomness of a string by the length of its shortest description. Thus, Kolmagorov randomness is an inherent property of individual strings. Unfortunately, this approach is non-constructive and is not applicable to pseudo-random number generation. Given their wide applicability, "pseudo-random number generators" have appeared with the first computers, before it was understood what properties they should satisfy. Evidently good pseudo-random sequences should have some of the statistical properties of truely random sequences. The statistical properties of linear congruential generators were extensively studied by Knuth [K]. This approach is empirical in nature and fails to yield general results of the form "for all practical purposes using the pseudo-random sequences is as good as using truely random ones". The fact that a pseudo-random number sequence passes some statistical tests, does not guarantee that it will pass a new test, i.e. that it is good for a future (untested) application. Recently, a constructive approach to randomness based on computation complexity has been initiated by Manuel Blum, Silvio Micali and Andy Yao [BM, Y]. Roughly speaking, a subset S of $\{0,1\}^n$ is *computationally random* if any statistical test that runs in polynomial time cannot distinguish strings randomly selected in S from strings randomly selected in $\{0,1\}^n$. Here "objects" are implicitly judged equal if they demonstrate the same "feasibly measurable" behavior. This approach is constructive both in the following two senses: First, one can test whether a subset is pseudorandom. Second, assuming the existence of one-way functions (i.e. functions that are easy to evaluate but hard to invert) one can specify a deterministic polynomial-time algorithm (*a generator*) that on input random k-bit long seeds, outputs k^c-bit strings ($c > 1$) undistinguishable, in polynomial time, from truly random k^c-long bit-strings. Essentially this means that any probabilistic polynomial time procedure gives the same results when its source of randomness is an unbiased coin or the output of such a generator. (Else the procedure would constitute an efficient statistical test that distinguishes the two). Under the thesis that all practical purposes correspond to polynomial-time procedures, the strings output by the generator are as good as truely random strings for all practical purposes.

Organization

In Section 2 we recall some basic notions from the theory of resource bounded computation and state our assumption concerning one-way functions. In Section 3, we present the fundamental definitions of "polynomially indistinguishable probability ensembles" and "pseudorandom ensembles". In Section 4 we show that, under the one-wayness assumption, pseudorandomness can be expanded. Section 5 consists of a discussion of the above notions and results. Further development and applications of the theory of pseudorandomness (e.g. pseudorandom functions) appear in Section 6.

2. BACKGROUND : RESOURCE – BOUNDED COMPUTATION

We begin this section by recalling the definitions of **P** and **BPP** – the complexity classes corresponding to deterministic and probabilistic polynomial-time computations. We continue by presenting the definition of one-way functions, which plays a central role in the construction of pseudorandom generators and in the general results concerning zero-knowledge proofs. The theory of resource bounded computations is developed in terms of asymptotic behaviour. Typically, we will consider the number of steps taken by an algorithm (i.e. a Turing machine) as a function of its input length, bound this function from above by a "smoother" function, and ask whether the bound is (or can be) a polynomial. This convention allows us to disregard special (short) inputs on which the machine may behave exceptionally good, and helps us concentrate on the "typical" behaviour of the machine.

2.1. Deterministic Polynomial-Time Computations

Traditionally in computer science, deterministic polynomial-time computations are associated with efficient computations. (Throughout the article, a *polynomial-time computation* means a computations in which the number of elementary computing steps is bounded by a polynomial in the length of the input.) This association stems from the acceptability of determinististic computing steps and polynomial-time computations as feasible in practice. The preference of deterministic computing steps (over non-deterministic ones) is self evident. Polynomial-time computations are advocated as efficient due to the moderate growing rate of polynomials and due to the correspondence between problems which are *known* to have "practically efficient" solutions and those *known* to have polynomial-time solutions. Deterministic polynomial-time computations are captured by the complexity class **P** (P stands for **P**olynomial). The complexity class **P** is defined as the set of languages satisfying for each $L \in$**P** there exists an algorithm A and a polynomial $p(\cdot)$ such that the following two conditions hold:

1) On input a bit string x ($x \in \{0,1\}^*$), algorithm A halts after at most $p(|x|)$ steps, where $|x|$ is the length of the string x.

2) On input x, algorithm A halts in an *accepting state* if and only if $x \in L$. (Otherwise it halts in a "rejecting state".)

2.2. Probabilistic Polynomial-Time Computations

Recent treads in computer science regard random computing steps as feasible. Following this approach, we consider computations which can be carried out by *Probabilistic* polynomial-time algorithms as modeling efficient computations. A probabilistic algorithm (or a "coin-tossing" algorithm) is one which (based on its current configuration) chooses its next move at random (with uniform probability distribution) among all possibilities. (In a deterministic algorithm, the next move is determined by the current configuration.) Without loss of generality, we assume that the number of possibilities (for the next configuration) is either 1 or 2. One can then view the algorithm as tossing an unbiased coin before each move and determining the next move using the outcome of the coin. On input x, the output of a probabilistic algorithm A is a random variable defined over the probability space of all possible internal coin tosses. Equivalently, probabilistic algorithms can be viewed as deterministic algorithms with two inputs: the ordinary input, and an auxiliary "random input". One then considers the probability distributions defined by fixing the first input and letting the auxiliary input assume all possible values with equal probability. In particular, the complexity class **BPP** (*BPP* stands for **B**ounded-away-error **P**robabilistic **P**olynomial-time) is defined as the set of languages such that for every $L \in BPP$ there exists a probabilistic polynomial-time algorithm A satisfying the following two conditions:

1) If $x \in L$ then $Prob(A(x) = 1) \geq \frac{2}{3}$.

2) If $x \notin L$ then $Prob(A(x) = 0) \geq \frac{2}{3}$.

It should be stressed that this definition is robust under substitution of $\frac{2}{3}$ by either $\frac{1}{2} + \frac{1}{p(|x|)}$ or $1 - 2^{-p(|x|)}$, where $p(\cdot)$ is an arbitrary positive polynomial. The following thesis captures the association of "efficient computation" with probabilistic polynomial-time computations.

Thesis: *BPP correspond to the set of computational problems which can be solved "efficiently".*

2.3. One-way Functions

It is generally believed that there exists (natural) problems, which demonstrate a gap between the complexity of finding a solution and the complexity of verifying its validity. For the results in this article we need to assume than there are problems which are hard on most (or at least on a "non-negligible" portion) of the instances. Furthermore, we assume that it is easy to generate hard instances together with a solution. This is formulated in terms of the infeasibility of inverting functions, which are easy to evaluate (in the forward direction).

Definition 1: A function $f : \{0,1\}^* \to \{0,1\}^*$ is called *one-way* if the following two conditions hold:

1) There exist a (deterministic) polynomial-time algorithm that on input x outputs $f(x)$.

2) For **any** probabilistic polynomial-time algorithm A', any constant $c > 0$, and sufficiently large n
$$Prob\left(A'(f(x), 1^n) \in f^{-1}(f(x))\right) < \frac{1}{n^c},$$
where the probability is taken over all x's of length n and the internal coin tosses of A', with uniform probability distribution.

Remark: The role of 1^k in the above definition is to allow machine A' to run in time polynomial in the length of the preimage it is supposed to find. (Otherwise, any function which shrinks the input more than by a polynomial amount will be considered one-way.)

Motivation to the notion of a negligible fraction: In the definition above, we have required that any machine trying to invert the function will succeed only on a "negligible" fraction of the inputs. We call *negligible* any function $\mu(\cdot)$ that remains smaller than 1 when multiplied by any polynomial (i.e., for every polynomial $p(\cdot)$ the limit of $\mu(n) \cdot p(n)$, when n grows to infinity, is 0). We ignore events which occur with negligible probability (as a function of the input length) since they are unlikely to occur even when repeating the experiment polynomially many times. On the other hand, events which occur with non-negligible (i.e., $1/p(n)$ for some polynomail p) probability will occur with almost certainty when repeating the experiment for a reasonable (i.e. polynomial) number of times. Thus, our notion of an "experiment" with a negligible success probability is robust (under polynomial number of repetitions of the experiment).

Motivation for considering infinitely many input lengths: The notion of a polynomial-time algorithm is meaningful only when considering infinitely many input lengths. (Otherwise one can always choose a polynomial which bounds the running time of a machine that halts on all inputs in some finite set.) Furthermore, for any instance length l, there exists a algorithm A_l which successfully inverts the function on all instances x of length l within $|x| + |f(x)|$ steps (algorithm A_l just incorporates in its transition function the inverses for all instances in this finite set). The same happens whenever we consider the inversion task for a finite set of instance lengths. Both technical difficulties are resolved when considering an infinite set of input lengths.

Assumption: *There exist one-way functions.* Furthermore, there exist one-way 1-1 and onto functions.

The following three number theoretic 1-1 and onto functions are widely believed to be one-way:

Ex1) **Modular Exponentiation**: In particular, let p be a prime and g be a primitive element of Z_p^* (the multiplicative group modulo p). Define $ME(p, g, x) = (p, g, y)$, where y is the result of reducing g^x modulo p. Inverting ME is known as the *Discrete Logarithm Problem.*

Ex2) **RSA**: Let p and q be primes, $N = p \cdot q$ and e be relatively prime to $\phi(N) = (p-1)\cdot(q-1)$. Define $RSA(N, e, x) = (N, e, y)$, where y equals $x^e \bmod N$.

Ex3) **Modular Squaring**: In particular, let p and q be primes both congruent to 3 mod 4, and $N = p \cdot q$. Define $MS(N, x) = (N, y)$, where y equals $x^2 \bmod N$. (To make this function one-to-one, restrict x to be a quadratic residue modulo N.) Inverting $MS(N, \cdot)$ is computationally equivalent to factoring N; that is, the problems are reducible to one another through probabilistic polynomial-time transformations.

The formulation of the above examples does not exactly fit the Definition 1, but things can be modified easily so they do. Alternatively, one may modify the constructions and theorems below.

3. DEFINITION OF PSEUDORANDOM DISTRIBUTIONS

A key definition in this approach is that of the infeasibility of distinguishing between two probability distributions. This behaviouristic definition, views distributions as equal if they cannot be told apart by any probabilistic polynomial-time test. Such a *test* receives as input a single string and outputs some statistics of the input. With no loss of generality, we may assume that the test outputs a single bit, which may be interpreted as a guess of the distribution from which the input was chosen. One considers the probability that, on input taken from the first distribution (resp. second distribution), the test outputs 1. If these two probabilities only differ by a negligible amount then the corresponding distributions are regarded as indistinguishable by this test.

Preliminaries (Probability Ensembles): A *probability distribution* is a function, π, from strings to non-negative reals such that $\sum_{\alpha \in \{0,1\}^*} \pi(\alpha) = 1$. A *probability ensemble* indexed by I is a sequence, $\Pi = \{\pi_i\}_{i \in I}$, of probability distributions. Throughout the entire article, we adopt the convention that the probability distributions in an ensemble assign non-zero probability only to strings of length polynomial in the length of the index of the distribution. Also, it suffices to consider $I = \{1\}^*$.

Motivation to defining ensembles: Probability ensembles are defined so that we can consider the asymptotic behaviour of arbitrary polynomial-time algorithms on inputs taken from a probability distribution.

Definition 2 (Polynomial Indistinguishability): Let $\Pi_1 = \{\pi_{1,i}\}_{i \in I}$ and $\Pi_2 = \{\pi_{2,i}\}_{i \in I}$ be two probability ensembles each indexed by elements of I. Let T be a probabilistic polynomial-time algorithm (hereafter called a *test*). The test gets two inputs: an index i and a string α. Denote by $p_1^T(i)$ the probability that, on input index i and a string α chosen according to the distribution $\pi_{1,i}$, the test T outputs 1 (i.e., $p_1^T(i) = \sum_\alpha \pi_{1,i}(\alpha) \cdot Prob(T(i, \alpha) = 1)$). Similarly, $p_2^T(i)$ denotes the probability that, on input i and a string chosen according to the distribution $\pi_{2,i}$, the test T outputs 1. We say that Π_1 and Π_2 are *polynomially indistinguishable* if for all probabilistic polynomial-time tests T, all constants $c > 0$ and all sufficiently large $i \in I$

$$|p_1^T(i) - p_2^T(i)| \quad |i|^{-c}.$$

Motivation to having the index as an auxiliary input to the test: In the above definition, when $I = \{1\}^*$, giving the index as auxiliary input to the test is not essential. We adopted this convention to make it consistent with other material (omitted here...).

An important special case of indistinguishable ensembles is that of probability ensembles which are polynomially indistinguishable from a uniform probability emsemble. These ensembles are called pseudorandom since, for all practical purposes, they are as good as truly unbiased coin tosses. This is clearly a behaviouristic point of view.

Definition 3 (Pseudorandom Distributions): Let $l : \{0,1\}^* \to N$ be a (*length*) function (mapping strings to integers), $\pi_{0,i}^l$ denote the uniform probability distribution on the set $\{0,1\}^{l(i)}$, and $\Pi_0^l = \{\pi_{0,i}^l\}_{i \in I}$. Let $\Pi_1 = \{\pi_{1,i}\}_{i \in I}$ be a probability ensemble indexed by I. We say that Π_1 is *pseudorandom* if it is polynomially indistinguishable from π_0^l, for some length function l.

Having defined pseudorandom ensembles, it is natural to ask whether such ensembles do exist. The answer is trivially affirmative, since the uniform ensemble is pseudorandom (being indistinguishable from itself!). However, this answer is of no interest. We would like to know whether ensembles which are not uniform, and furthermore are not statistically close to uniform, can be pseudorandom. Furthermore, can such ensembles be constructed using less coin tosses than the length of the strings in their support? The answer to both questions is affirmative. Namely,

Theorem 1: There exist pseudorandom ensembles which are not statistically close to a uniform ensemble. In particular, there exists a pseudorandom ensemble $\Pi = \{\pi_i\}_{i \in I}$ such that the support of π_i consists of $2^{|i|}$ strings, each of length $2 \cdot |i|$. Furthermore, there exists an (exponential-time) probabilistic algorithm that on input i tosses $|i|$ coins and outputs strings with distribution π_i.

Proof's Idea: By a counting argument. ■

4. ON THE EXPANDABILITY OF PSEUDORANDOM DISTRIBUTIONS

We have concluded the previous section by arguing that pseudorandom sequences which are very sparse can be constructed using less coin tosses than their length. However, this construction was not computational efficient and thus could not be applied in practice. The real question is whether such an expansion of randomness can be carried out efficiently. A key definition capturing this question follows.

Definition 4 (Pseudorandom Generator): Let $p(\cdot)$ be a polynomial satisfying $p(n) \geq n + 1$. Let G be a deterministic polynomial-time algorithm that on input any n-bit string, outputs a string of length $p(n)$. Let $\underline{n}$ denote the unary encoding of the integer n. We say that G is a *pseudorandom* *generator* if the probability ensemble defined by it is pseudorandom. Here, the ensemble defined by G is $\{G_{\underline{n}}\}$ where a string y has probability $m \cdot 2^{-n}$ in the distribution $G_{\underline{n}}$ if there are exactly m seeds of length n such that feeding each of them to G yields the output y.

Motivation to the unary encoding of the length: The length of the seed to G (i.e. n) is encoded in unary so that the strings in the support of $G_{\underline{n}}$ have length polynomial in n ($= |\underline{n}|$). Now, we can formally state the fundamental question of *whether pseudorandom generators do exist.* We will see that, under the assumption that one-way 1-1 and onto functions exist, the answer is **yes**. The following definitions and results are used in order to prove this implication. In particular, the equivalence of Definition 3 and Definition 5 plays an important role in proving the pseudorandomness of the construction presented below. Definition 5 concerns the feasibility of predicting the next bit in a string, which is taken from some distribution. The predictor is given only a prefix of the string. The question is whether there exists an efficient predictor which succeeds with probability non-negligibly greater than $\frac{1}{2}$.

Definition 5 (Unpredictability): Let $\Pi_1 = \{\pi_{1,i}\}_{i \in I}$ be a probability ensemble indexed by I. Let A be a probabilistic polynomial-time algorithm that on inputs i and y outputs a single bit (called the *guess*). Let $bit(\alpha, r)$ denote the r-th bit of the string α, and $pref(\alpha, r)$ denote the prefix consisting of the first r bits of the string α (i.e. $pref(\alpha, r) = bit(\alpha, 1)bit(\alpha, 2) \cdots bit(\alpha, r)$). We say that the algorithm A *predicts the next bit* of Π_1 if for some $c > 0$ and infinitely many i's

$$Prob\left(M(i, pref(\alpha, r)) = bit(\alpha, r+1)\right) \geq \tfrac{1}{2} + |i|^{-c},$$

where the probability space is that of the string α chosen according to $p_{1,i}$, the integer r chosen at random with uniform distribution in $\{0, 1, ..., |\alpha| - 1\}$ and the internal coin tosses of M. We say that Π_1 is *unpredictable* if there exist **no** probabilistic polynomial-time algorithm A which predicts the next bit of Π_1. Definition 5 can be viewed as a special case of Definition 3. Any predictor can be easily converted into a test which outputs 1 if and only if the guess of the predictor is correct. The resulting test will distinguish an ensemble from the uniform ensemble if and only if the original predictor's guesses are non-negligibly better than "random". Interestingly, the special case is not less powerful. Namely, each successful distinguisher can be converted into a successful predictor.

Theorem 2: *Let Π_1 be a probability ensemble. Then Π_1 is pseudorandom if and only if it is unpredictable.*

Proof's Idea: Assume that T is a test which distinguishes $\pi_{1,i}$ from the uniform distribution. We consider the behaviour of T when fed with strings taken from the *hybrid* distributions $H_i^{(j)}$, where $H_i^{(j)}$ is defined as the distribution resulting by taking the first j bits of a string chosen from $\pi_{1,i}$ and letting the other bits be uniformly distributed. There must be two adjacent hybrids, $H_i^{(j)}$ and $H_i^{(j+1)}$, which are distinguishable by T. The $j+1$st bit is predicted using this "gap". ■

The notion of a hard-core predicate (presented below) plays a central role in the construction of pseudorandom generators. Intuitively, a hard-core of a function f is a predicate ($b(x)$) which is easy to evaluate (on input x) but hard to even approximate when given the value of the function ($f(x)$). Recall that f is one-way if it is easy to evaluate (i.e. compute $f(x)$ from x) but hard to invert (i.e. compute x from $f(x)$). Thus, the hard-core maintains in a strong sense both the easyness (in the forward direction) and the hardness (in the backward direction) of the function.

Definition 6 (Hard-core Predicate): Let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ and $b : \{0,1\}^* \rightarrow \{0,1\}$. The predicate b is said to be *a hard-core* of the function f if the following two conditions hold

1) There is a deterministic polynomial-time algorithm that on input x returns $b(x)$.
2) There is **no** probabilistic polynomial-time algorithm A' such that for some $c > 0$ and infinitely many n

$$Prob\,(A'(f(x)) = b(x)) \geq 1/2 + n^{-c},$$

where the probability is taken over all possible choices of $x \in \{0,1\}^n$ and the internal coin tosses of A' with uniform probability distribution.

Clearly, if the predicate b is a hard-core of the 1-1 and onto function f then f is hard to invert. Assuming that either of the functions presented in subsection 2.4 is one-way, predicates which constitutes corresponding hard-core can be presented. For example, the least significant bit is a hard-core of RSA (i.e., given $RSA(N, e, x)$ one cannot efficiently predict the least significant bit of x). In fact, every one-way function f can be "transformed" into a one-way function f' with a corresponding hard-core predicate b'. Thus, unpredictability and computational difficulty play dual roles.

Theorem 3: *If there exist one-way functions (resp. one-way 1-1 and onto functions) then there exist one-way functions (resp. one-way 1-1 and onto functions) with a hard-core predicate.*

Proof's Idea: The proof uses the observation that if f is one-way then there must be a bit in its argument x that cannot be efficiently predicted from $f(x)$ with success probability greater than $1 - 1/|x|$. (Otherwise, with constant probability, all the bits of the argument can be predicted correctly and the argument can be retrieved.) Let $b(i, x)$ denote the ith bit of x. For $|x_1| = |x_2| = \cdots = |x_{n^3}| = n$, define

$$f'(x_1, x_2, ..., x_{n^3}) = f(x_1)f(x_2)\cdots f(x_{n^3}),$$
$$b'(x_1, x_2, ..., x_{n^3}) = \left(\sum_{i=1}^{n} \sum_{j=1}^{n^2} b(i, x_{(i-1)\cdot n^2+j}) mod\, 2 \right).$$

It can be shown that the predicate b' is a hard-core of f'. The proof *does not* reduce to showing that a (sufficiently long) sequence of biased and independent 0-1 random variables has sum mod 2 which is almost unbiased (since the prediction errors on the various predicates are not random variables)! Reproducing the actual proof is beyond the scope of this article. ■

Having a one-way 1-1 and onto function with a hard-core predicate suffices for the following construction of pseudorandom generators.

Construction 1: Let f be a one-way 1-1 and onto function and b a hard-core predicate of f. We define the following polynomial-time algorithm G. On input x, algorithm G computes the bits $b_i = b(f^{(i)}(x))$, where $1 \le i \le 2|x|$ and $f^{(i)}$ denotes the function f iteratively applied i times. Machine G outputs $b_{2|x|} \cdots b_2 b_1$.

Lemma 1: Let f, b and G be as in Construction 1. Then $\{G_{\underline{n}}\}$ defined as in Definition 4 is unpredictable.

Proof's Idea: An efficient predictor of the sequence defined above can be easily converted into a algorithm M that on input $f(x)$ guesses $b(x)$ with success probability greater than $1/2$. On input $f(x)$, algorithm M computes the sequence $b(f^{(k)}(x)), ..., b(f^{(2)}(x)), b(f(x))$ and obtains a prediction for $b(x)$. ■

Combining Theorem 3, Lemma 1 and Theorem 2, we get

Theorem 4: *If there exist one-way 1-1 and onto functions then there exist pseudorandom generators.*

We conclude that feeding a pseudorandom generators with seeds taken from a uniform distribution (over $\{0,1\}^n$), yields a pseudorandom distribution. The following theorem states that feeding a pseudorandom generator with seeds taken from a pseudorandom distribution also yields a pseudorandom distribution over longer strings.

Theorem 5: Suppose that $\Pi_1 = \{\pi_{1,i}\}_{i \in I}$ is a pseudorandom ensemble, and G is a pseudorandom generator. Then the ensemble $\Pi_2 = \{\pi_{2,i}\}_{i \in I}$, where $\pi_{2,i}$ is defined by feeding G with inputs according to the distribution $\pi_{1,i}$, is also pseudorandom.

Proof's Idea: Assume to the contrary that there exists a (polynomial-time) test T distinguishing between Π_2 and the uniform distribution. Then at least one of the following two statements hold:

1) The test T also distinguishes $\{G_{\underline{n}}\}$ from the uniform distribution (in contradiction to G being a pseudorandom generator).

2) The test T can be modified into a test T' (which first applies G to the tested string and then runs T on the result) so that T' distinguish Π_1 from the uniform distribution (thus contradicting the hypothesis that Π_1 is pseudorandom). ■

5. DISCUSSION

Before presenting further extensions and applications of the above approach to randomness, let us discuss several conceptual aspects.

Behavioristic versus Ontologic

The behaviouristic nature of the above approach to randomness is best demonstrated by confronting this approach with the Kolmogorov-Chaitin approach to randomness. Loosely speaking, a string is *Kolmogorov-random* if its length equals the length of the shortest program producing it. This shortest program may be considered the "true explanation" to the phenomenon described by the string. A Kolmogorov-random string is thus a string which does not have a substantially simpler (i.e. shorter) explanation than itself. Considering the simplest explanation of a phenomenon is certainly an ontologic approach. In contrast, considering the effect of phenomena on certain objects, as underlying the definition of pseudorandomness (above), is a behaviouristic approach. Furthermore, assuming the existence of one-way 1-1 and onto functions, there exist probability distributions which are not uniform (and are not even statistically close to a uniform distribution) that nevertheless are indistinguishable from a uniform distribution (by any efficient method). Thus, distributions which are ontologically very different, are considered equivalent by the behaviouristic point of view taken in the definitions above.

A Relativistic View of Randomness

Pseudorandomness is defined above in terms of its observer. It is a distribution which cannot be told apart from a uniform distribution by any efficient (i.e. polynomial-time) observer. Thus, pseudorandomness is subjective to the abilities of the observer. To illustrate this point consider the following *mental experiment.*

> Alice and Bob want to play "head or tail" in one of the following four ways. In all of them Alice flips an unbiased coin and Bob is asked to guess its outcome before the coin rests on the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first way, Bob has to announce his guess before Alice flips the coin. Clearly, in this way Bob wins with probability 1/2. In the second way, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability 1/2. The third way is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth way, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve his guess of the outcome of the coin substantially.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Pseudorandom ensembles are unpredictable by probabilistic polynomial-time algorithms (associated with feasible computations), but may be predictable by infinitely powerful machines (not at our disposal!).

Effectiveness and Applicability

Another interesting property of the above approach to randomness is that it is effective in the following two senses: First, one may construct an efficient (universal) test that distinguishes pseudorandom distributions from ones which are not pseudorandom. In contrast, the problem of determining whether a string is Kolmogorov-random is undecidable. Second, assuming the existence of one-way 1-1 and onto functions, long pseudorandom strings can be efficiently and deterministically generated from much shorter pseudorandom strings. Clearly, this cannot be the case with Kolmogorov-random strings. The existence of pseudorandom generators has applications to the construction of efficient probabilistic algorithms. Such algorithms maintain the same performance when substituting their internal coin tosses by pseudorandom sequences. Thus, for every constant $\epsilon > 0$, the number of truly random bits required in a polynomial-time computation on input x can be decreased (from $poly(|x|)$) to $|x|^{\epsilon}$.

Randomness and Computational Difficulty

Randomness and computational difficulty play dual roles. This was pointed out already when discussing one-way functions and hard-core predicates. The relationship between pseudorandom generators and one-way computations is even a better illustration of this point. We have shown above that the existence of one-way 1-1 and onto functions implies the existence of pseudorandom generators. On the other hand, one can readily verify that any pseudorandom generator constitutes a one-way function.

6. FURTHER EXTENSIONS AND APPLICATIONS

6.1. Pseudorandom Functions or Experimenting with the Random Source

In the previous subsection we have (implicitly) modelled phenomena as single events (bit strings). This model suffices for describing phenomena in which the observer is passive: he can only record the events which occur. A more powerful model allows the observer to conduct experiments. Namely, "feed" the phenomenon with some values and measure the events which correspond to these values. Modelling a phenomenon as a function from events to events (or as a function from environment values to actions) is thus natural and useful. As in the previous subsections, we will present definitions for a pair of indistinguishable phenomena, a pseudorandom phenomenon and a generator of the latter. In other words, we will present definitions for indistinguishability of functions, pseudorandom functions, and pseudorandom function generators. For our definition of indistinguishable function ensembles we consider Turing machines (i.e. algorithms) with oracles. These machines are able, in addition to the traditional computing steps, to make *oracle queries*: place a string on a special tape and read an "answer" in the next step. Loosely speaking, we will say that two function ensembles are indistinguishable if any polynomial-time oracle Turing machine cannot distinguish the case that its oracle is a function taken from the first ensemble and the case that the oracle is a function taken from the second.

Definition 7 (Indistinguishability of Functions): Let $F_1 = \{F_{1,i}\}_{i \in I}$ and $F_2 = \{F_{2,i}\}_{i \in I}$ be two function ensembles, where $F_{j,i}$ is a probability distribution on the functions $f : \{0,1\}^{|i|} \to \{0,1\}$. We say that F_1 and F_2 are *polynomially indistinguishable* if for every probabilistic polynomial-time oracle machine M, every constant $c > 0$ and all sufficiently large $i \in I$

$$|p_1^M(i) - p_2^M(i)| < |i|^{-c},$$

where $p_j^M(i)$ is the probability that M outputs 1 on input i when querying an oracle randomly chosen from the distribution $F_{j,i}$.

Definition 8 (Pseudorandom Functions and Function Generators): The function ensemble $F = \{F_i\}_{i \in I}$ is *pseudorandom* if it is polynomially indistinguishable from the ensemble $H = \{H_i\}_{i \in I}$, where H_i is the uniform probability distribution on the set of functions $f : \{0,1\}^{|i|} \to \{0,1\}$.
We say that $F = \{F_{\underline{n}}\}$ is a *pseudorandom function generator* if the following three conditions hold:

1) There exists a probabilistic polynomial-time machine M_1 that, on input $\underline{n}$, randomly selects a function f from the distribution $F_{\underline{n}}$, and outputs a (succinct) description of f (denoted $\tilde{f}$).

2) There exists a (deterministic) polynomial-time machine M_2 that, on input $\tilde{f}$ (a description of $f : \{0,1\}^n \to \{0,1\}$) and a string x ($\in \{0,1\}^n$), outputs $f(x)$. That is, $M_2(\tilde{f}, x) = f(x)$.

3) The ensemble F is pseudorandom.

Similar definitions apply to function ensembles consisting of distributions F_i on functions mapping $\{0,1\}^{|i|}$ to $\{0,1\}^{|i|}$. Furthermore, one can easily transform ensembles of the first kind to ones of the second type, and vice versa. As in subsection 3.2, we now ask whether there exist non-trivial ensembles of pseudorandom functions, and furthermore whether such ensembles can be efficiently generated. It turns out that this question reduces to the question handled in subsection 3.2. Namely,

Theorem 6: *Pseudorandom function generators exist if and only if pseudorandom generators exist.*

Proof's Idea: The "only if" direction of Theorem 6 is easy. The generator first uses M_1 to get an $\tilde{f}$ and next uses M_2 to evaluate $f(1)$, $f(2), \ldots$ The "if" direction of the Theorem also has a constructive proof. The construction proceeds in two steps: First one uses an arbitrary pseudorandom generator to construct a pseudorandom generator G that doubles the length of its input. Next, G is used to construct a pseudorandom function in the following manner. Let $G_0(x)$ denote the first $|x|$ bits output by G on input x, and $G_1(x)$ denote the last $|x|$ bits output by G on input x. Extend the above notation so that for every bit σ and bit string α, $G_{\alpha\sigma}(x) = G_\alpha(G_\sigma(x))$. Now, let $f_x(y) = G_y(x)$, and $F_{\underline{n}}$ is the distribution obtained by picking x uniformly among all n bit strings and using the resulting function f_x. It can be shown that F so defined is a pseudorandom function generator. (It is interesting to note that this is not the case if we let $f_x(y) = G_x(y)$.) ■

Further Discussion It is interesting to point out the analogy between the above definition of pseudorandom functions and Turing's famous "test of intelligence". (In Turing's test of intelligence, one is interacting arbitrarily with an unknown entity which is either a human or a machine. The machine is said to be (pseudo)intelligent if the tester cannot distinguish the two cases.) In both settings one interacts with an unknown function in order to latter determine the "nature" of this function. Failure to determine the "true nature" is interpreted as a proof that the difference in nature is of no importance (as far as functionality goes...). Pseudorandom functions can not be predicted, even not in the following weak sense: any probabilistic polynomial-time oracle Turing machine cannot predict the value of the oracle on an unasked query better than 50-50, when the oracle is a pseudorandom function. This resembles the following quotation of Turing:

> *I have set up on a Manchester computer a small programme using only 1000 units of storage, whereby the machine supplied with one sixteen figure number replies with another within two seconds. I would defy anyone to learn from these replies sufficient about the programme to be able to predict any replies to untried values.*

6.2. Applications to Cryptography

The most obvious application of pseudorandomness to cryptography is making *one-time pads* a feasible and secure encryption method. One-time pads are the simplest and safest private-key cryptosystem. A cleartext is encrypted by XORing its bits with the currently initial segment of the (*randomly selected*) key, and the resulting ciphertext is decrypted by XORing its bits with the very segment of the key. Each segment of the key is deleted after use, and thus no information about the cleartext can be extracted from the ciphertext. The drawback of one-time pads is that the length of the key in use must equals or even exceed the length of the messages sent. Namely, in order to *secretly* pass a message of length l one must exchange secretly another message of length

l. This is not satisfactory both from a theoretical and practical point of view, since the aim is to achieve high level of security in a much lower "cost". In practice, "pseudorandom sequences" are used instead of the randomly selected key of the one-time pad, *but security can no longer be asserted.* Assuming the existence of pseudorandom bit generators (in the sense discussed in section 3.2), one can replace the key of the one-way pad by a pseudorandom sequence and **prove** that the resulting cryptosystem is secure in the following sense: *whatever can be efficiently computed from the ciphertext can be efficiently computed without it.* In other words, as far as polynomial-time computations are concerned, no information about the cleartext is *revealed* from the ciphertext. Other applications of pseudorandomness to Cryptography use the construction of pseudorandom functions (Theorem 6, section 3.4). For example, it is possible to produce unforgeable *message authentication tags* and *time-stamps.* Assume two parties A and B, sharing a secret key, communicate over a channel tampered by an adversary C. The adversary may inject messages on the channel. The parties would like to be able to verify that a message has arrived from their counterpart, and not from the adversary. It is suggested that in order to authenticate a message M, party A just applies the pseudorandom function f to M, and sends $f(M)$ as the authentication tag of M. Party B may then verify the validity of this authentication tag, being confident that the message has been sent by A (and not injected by C). We stress that if f is a pseudorandom function then the above scheme is **provably** secure in the following sense: *even if C gets polynomially many authentication tags to messages of his choice he cannot produce in polynomial-time an authentication tag to any other message.*

6.3. Necessary and Sufficient Conditions of the Existence of Pseudorandom Generators

The condition in Theorem 4 can be relaxed so to derive a necessary and sufficient condition for the existence of pseudorandom generators. Essentially, one requires a function f which is "one-way on its iterates"; namely, that f is hard to invert on the distribution obtained by applying the function iteratively $k^{1.5}$ times, where k is the length of the argument. Clearly, any one-way permutation is one-way on its iterates.

Theorem $4'$: *Functions which are one-way on their iterates exist if and only if pseudorandom generators exist.*

Proof's Idea: For the $\rightarrow$ direction one should carefully modify the proof of Theorem 4. The $\leftarrow$ direction follows by slightly modifying the pseudorandom generator. ■

We believe that the reader will not find the above Theorem $4'$ satisfactory, and urge him to prove our conjecture that the above two conditions are in fact equivalent to the mere existence of arbitrary one-way functions. Partial progress is reported in [GKL].

7. CONCLUSIONS

The fact that pseudorandom generators and functions exist under a reasonable complexity theoretic assumption (i.e. the existence of one-way 1-1 and onto functions), must be considered at least a plausibility argument. Thus, every reasoning overruling the existence of such generators must incorporate a demonstration that one-way 1-1 and onto functions do not exist. The *possible existence* of pseudorandom generators does not allow us to consider "unbounded" random behaviour as necessarily arising from an "unbounded" source of randomness, since a pseudorandom generator may expand a "bounded" amount of randomness to an "unbounded" amount of pseudorandomness. Furthermore, the possible existence of pseudorandom functions implies that a small amount of randomness suffices in order to efficiently determine a random mapping from huge sets into huge sets. All the above was discovered through a behaviouristic approach to the notion of randomness. We believe that a behaviouristic approach is justified when studying computing devices, as much as it is unjustified when studying "thinking beings".

ACKNOWLEDGEMENTS

First of all, I would like to thank two remarkable people who had a tremendous influence on my professional development. Shimon Even introduced me into theoretical computer science and closely guided me in my first steps. Silvio Micali led my way in the evolving foundations of cryptography and shared with me his efforts of further developing them. Next, I would like to thank Benny Chor for his indispensable contribution to our joint research, and for the excitement and pleasure I had when collaborating with him. Special thanks to Leonid Levin for many interesting discussions. Finally, I would like to thank Hugo Krawczyk for carefully reading an earlier version of the manuscript, pointing out some errors, and suggesting several improvements.

BIBLIOGRAPHIC NOTES

For background on Computational Complexity consult an appropriate textbook such as [HU, ch. 12-13] and [GJ]. The notion of one-way functions was first suggested in [DH], and the most famous candidate is due to [RSA]. A 1-1 function which is one-way, unless factoring is easy appears in [R]. Definition 1 (*one-way functions*), however, is a weaker form and is due to [Y]. A special case of Definition 2 (*indistinguishability*) first appeared in [GM], the general case is from [Y]. Definitions 3 and 4 (*pseudorandomness*) are due to [Y], while Definition 5 (*unpredictability*) appears in [BM]. Theorem 2 (*equivalence of Def's 3 and 5*) is implicit in [Y]. Definition 6 (*hard-core predicate*), Construction 1 (*pseudorandom generator based on a hard-core predicate*) and Lemma 1 appear in [BM]. Theorem 3 (*existence of hard-core predicates assuming one-way 1-1 functions*) is implicit in [Y], where a sketch of the proof of Theorem 4 (*pseudorandom generator based on one-way 1-1 functions*) appears. A finer analysis, which leads to Theorem 4′ (*a necessary and sufficient condition for the existence of pseudorandom generators*, appears in [L]. Predicates which are hard-core of the particular number theoretic functions mentioned in section 2.4, appear in [BM] and [ACGS]. Definitions 7 and 8 (*pseudorandom functions*) and Theorem 6 (*pseudorandom generators imply pseudorandom function generators*) appear in [GGM]. Further developments appear in [LR].

REFERENCES

[ACGS] Alexi, W., B. Chor, O. Goldreich and C.P. Schnorr, "RSA and Rabin Functions: Certain Parts Are As Hard As the Whole", *Proc. 25th IEEE Symp. on Foundation of Computer Science*, 1984, pp. 449-457, (to appear in *SIAM J. Computing*).

[BM] Blum, M., and Micali, S., "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM Jour. on Computing*, Vol. 13, 1984, pp. 850-864.

[DH] Diffie, W., and M. E. Hellman, "New Directions in Cryptography", *IEEE transactions on Info. Theory*, IT-22 (Nov. 1976), pp. 644-654

[GJ] Garey, M.R., and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How to Construct Random Functions", *Jour. of ACM*, Vol. 33, No. 4, 1986, pp. 792-807.

[GKL] Goldreich, O., H. Krawczyk, and M. Luby, "On the Existence of Pseudorandom Generators", priprint, 1987.

[GM] Goldwasser, S., and S. Micali, "Probabilistic Encryption", *JCSS*, Vol. 28, No. 2, 1984, pp. 270-299.

[HU] Hopcroft, J.E., and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Co., 1979.

[L] Levin, L.A. "One-Way Function and Pseudorandom Generators", *Proc. 17th ACM Symp. on Theory of Computing*, 1985, pp. 363-365.

[LR] Luby, M., and C. Rackoff, "Pseudo Random Permutation Generators and DES", *Proc. 18th ACM Symp. on Theory of Computing*, 1986, pp. 356-363.

[R] Rabin, M.O. "Digitalized Signatures and Public Key Functions as Intractable as Factoring", MIT/LCS/TR-212, 1979.

[RSA] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. ACM*, Vol. 21, Feb. 1978, pp. 120-126

[Y] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.

APPENDIX: ALTERNATIVE PRESENTATION FOR SECTION 4

Technically speaking, the proof of Theorem 3 can be simplified by replacing Yao's XOR Lemma with an alternative (and efficient) construction of Goldreich and Levin [2]: For every one-way function f, the inner-product mod 2 of x and r is a hardcore of the function $f'(x,r) = (f(x), r)$. (By the way, an accessible exposition of the proof of Yao's XOR Lemma are now available (cf., [3]).) Furthermore, I currently prefer an alternative presentation, as in [1], which proceeds as follows: First, one shows (directly) that, for any one-way 1-1 onto function f and hardcore b, the function $G(s) = f(s)b(s)$ is a pseudorandom generator. Next, one shows how to transform any pseudorandom generator into one which doubles the length of its input. Finally, we comment that it has been shown that the existence of any one-way function implies the existence of a pseudorandom generator [4].

References

[1] O. Goldreich. *Foundation of Cryptography – Fragments of a Book.* February 1995. Available from `http://theory.lcs.mit.edu/~oded/frag.html`.

[2] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st STOC*, pages 25–32, 1989.

[3] O. Goldreich, N. Nisan and A. Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[4] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. Construction of Pseudorandom Generator from any One-Way Function. To appear in *SICOMP.* Preliminary versions by Impagliazzo et. al. in *21st STOC* (1989) and Håstad in *22nd STOC* (1990).